

# A SCALABLE GENERATIVE GRAPH MODEL WITH COMMUNITY STRUCTURE\*

TAMARA G. KOLDA<sup>†</sup>, ALI PINAR<sup>†</sup>, TODD PLANTENGA<sup>†</sup>, AND C. SESHADHRI<sup>†</sup>

**Abstract.** Network data is ubiquitous and growing, yet we lack realistic generative network models that can be calibrated to match real-world data. The recently proposed Block Two-Level Erdős-Rényi (BTER) model can be tuned to capture two fundamental properties: degree distribution and clustering coefficients. The latter is particularly important for reproducing graphs with community structure, such as social networks. In this paper, we compare BTER to other scalable models and show that it gives a better fit to real data. We provide a scalable implementation that requires only  $O(d_{\max})$  storage where  $d_{\max}$  is the maximum number of neighbors for a single node. The generator is trivially parallelizable, and we show results for a Hadoop implementation for a modeling a real-world web graph with over 4.6 billion edges. We propose that the BTER model can be used as a graph generator for benchmarking purposes and provide idealized degree distributions and clustering coefficient profiles that can be tuned for user specifications.

**Key words.** graph generator, network data, block two-level Erdős-Rényi (BTER) model, large-scale graph benchmarks

**1. Introduction.** Unprecedented amounts of network interaction data are now available from online social networks (FaceBook, Twitter), massive multi-player online games (World of Warcraft, Everquest), computer-to-computer communications, financial transactions, instant messaging, and more. As a result, models, algorithms, software, and hardware for large-scale graph analysis are struggling to keep pace with increasing demands for scalability and relevance. For instance, we lack models that provide a realistic baseline for statistical analysis such as anomaly detection. Additionally, a major obstacle to working in the field of network analysis is that data is naturally restricted due to a combination of security and privacy concerns. For these reasons, we need scalable generative models for networks.

Ideally, a generative model can be calibrated to match real world data. For the purposes of this paper, we consider the two most fundamental properties of graphs: the degree distribution and the clustering coefficients [40]. Let  $G = (V, E)$  be an undirected, unweighted graph on vertices  $V$  defined by edges in  $E$ , let  $n = |V|$  denote the number of nodes, and let  $m = |E|$  denote the number of edges.

In most real-world networks representing interaction data, there are a few nodes with high degrees and many nodes with low degrees, with a smooth transition between. In other words, the degree distribution is heavy-tailed, and this feature has long been considered as the critical feature that distinguishes real networks from arbitrary sparse networks [2, 12, 35]. A realistic generative model should be able to reproduce the *degree distribution*, which specifies the number of nodes of each degree  $\{n_d\}$ .

However, the structure prescribed by the degree distribution is only part of the story; real-world graphs have an abundance of triangles. In social networks, triangles indicate social cohesion, and it is likely that two people with a common friend are much more likely to be friends themselves. Figure 1.1 shows a *wedge* centered at node 1, i.e., the path 2-1-3 is a wedge. If edge 2-3 exists, then the wedge is closed (forms

---

\*This work was funded by the GRAPHS Program at DARPA and by the Applied Mathematics Program at the U.S. Department of Energy. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

<sup>†</sup>Sandia National Laboratories, Livermore, CA. Email: {tgkolda, apinar, tplante, scomand}@sandia.gov

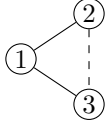


Fig. 1.1: Wedge centered at node 1.

a triangle); otherwise it is open. Note that every triangle corresponds to three closed wedges, so the number of closed wedges is three times the number of triangles. To measure this, we define the *global clustering coefficient* (GCC) [41, 3] as

$$c = \frac{\# \text{ of closed wedges}}{\# \text{ wedges}}.$$

We are specifically interested in *clustering coefficient per degree* [41], defined as

$$c_d = \frac{\# \text{ of closed wedges centered at a node of degree } d}{\# \text{ wedges centered at a node of degree } d}.$$

The clustering coefficients of real-world graphs are much higher than those of random graphs with the same degree distribution [17]. It is known that real-world graphs have significant clustering coefficients and that these are important to community structure. Nonetheless, most generative models fail to match clustering coefficients of real-world graphs [34].

**1.1. Contributions.** The Block Two-Level Erdős–Rényi (BTER) [36] has been proposed as a model that can be tuned to capture both the degree distribution and degree-wise clustering coefficients for real-world networks. The goal of this paper is to focus on the *implementation and scalability* of this model. Hence, this paper makes the following contributions:

- We describe a scalable implementation of the BTER generative graph model that uses efficient data structures. Our reference implementation requires working storage of at most  $10 \cdot d_{\max}$  values where  $d_{\max} \ll n$  is the largest degree. The generation cost per edge is  $O(\log d_{\max})$ . Our approach generates all edges independently and thus can be easily parallelized. Moreover, the edges can be generated in an arbitrary order, so the BTER generative model can also be used in streaming scenarios.
- We demonstrate that BTER can accurately recreate the degree distributions and triangle behavior of large real-world graphs. We show results for several example graphs from the Laboratory for Web Algorithms [43], including a graph with over 130 million nodes and 4.6 billion edges. We also compare BTER to competing methods on a pair of smaller graphs.
- Finally, we propose BTER as a standard graph generator for benchmarking purposes. Since BTER can work with arbitrary degree distributions, we propose an “ideal” degree distribution: discrete generalized log-normal (DGLN). This is a two-parameter model that can easily match a desired maximum possible degree (an absolute bound) and a desired average degree. We also propose a semilog-linear model for clustering coefficients.

We note that there is no fitting step for BTER — it takes the target degree distribution and clustering coefficients per degree directly as input. The clustering coefficients can be expensive to compute, but we have recently proposed a sampling method that scales to very large graphs [38, 20].

**1.2. Related Work.** Since the goal of this paper is to focus on the implementation and scalability of BTER, we limit our discussion to the most salient related models. A more thorough discussion of related work can be found in the original paper on BTER [36].

The majority of graph models add edges one at a time in a way that each random edge influences the formation of future edges, making them inherently unscalable. The classic example is Preferential Attachment [2], but there are a variety of related models, e.g., [21, 24]. These models are more focused on capturing qualitative properties of graphs and typically are difficult to match to real-world data [34]. Perhaps the most relevant is [19], which creates a graph with power law degree distribution and then “rewires” it to improve the clustering coefficients. Another related model, the clustering model proposed Newman [31], assigns “individuals” to “groups” (a bipartite graph with individual and group nodes) and then creates a graph of connections between individuals by assigning connection probabilities to each group; in other words, each group is modeled as an Erdős–Rényi graph.

A widely used model for modeling large-scale graphs is the Stochastic Kronecker Graph (SKG) model, also known as R-MAT [8, 23]. The generation process is easily parallelized and can scale to very large graphs. Notably, SKG has been selected as the generator for the Graph 500 Supercomputer Benchmark [42] and has been used in a variety of other studies [13, 26, 29, 28, 18, 14, 27]. Unfortunately, SKG has some drawbacks. (1) SKG can be extremely expensive to fit to real data (using KronFit), and even then the fit is imperfect [23]. (2) It can generate only lognormal tails (after suitable addition of random noise) [39, 37], limiting the degree distributions that it can capture. (3) Most importantly, SKG rarely closes wedges so the clustering coefficients are much smaller than what is produced in real data [34, 20].

Another model of relevance is the Chung-Lu (CL) model [10, 11, 1]. It is very similar to the edge-configuration model of Newman et al. [30]. Let  $d_i$  denote the *desired degree* for node  $i$ . In the CL model, the probability of an edge is proportional to the degrees of its endpoints, i.e., the probability of edge  $(i, j)$  is  $\propto d_i d_j$ . Edges can be generated independently by picking endpoints proportional to their desired degrees. If all degrees are the same, CL reduces to the well-known Erdős–Rényi model [15]. The CL model is often used as a null model; for example, it is the basis of the modularity metric [32]. Graphs generated by the CL model and the SKG model are, in fact, very similar [33]. The advantage of the CL model is that it can be better tuned to real-world degree distributions. The disadvantage of the model is that, like SKG, it rarely closes wedges. CL is a special case of BTER that skips Phase 1 (see §2), and the CL construction is a very important part of BTER.

## 2. BTER Generative Graph Model.

**2.1. The BTER Model.** Our earlier work argued that a graph with a heavy-tailed degree distribution and high clustering coefficients must contain Erdős–Rényi blocks of densely connected vertices; moreover, the distribution of the sizes of those groups follows the same type of distribution of the degrees (e.g., power law) [36]. Based on this premise, the BTER model [36] organizes nodes into *affinity blocks* such that nodes within the same affinity block have a much higher chance of being connected than nodes at random, but BTER also behaves like the CL model in that it is able to match an arbitrary degree distribution.

We first give a high level description of a serial version of BTER [36]. The BTER model requires two user-specified inputs: (1) desired degree distribution, denoted by  $\{n_d\}$  where  $n_d$  is the number of nodes of degree  $d$ ; and (2) clustering coefficient by

degree, denoted by  $\{c_d\}$  where  $c_d$  is the desired mean clustering coefficient for nodes of degree  $d$ . (We note that the original description in [36] did not take the original clustering coefficients but rather a function to determine the connectivity.) The desired number of nodes and edges are  $n = \sum_d n_d$  and  $m = \frac{1}{2} \sum_d d \cdot n_d$ , respectively. There are three main steps to the graph generation, as described below. The steps are depicted in Figure 2.1.

*Preprocessing.* Imagine starting with  $n$  isolated vertices. Each vertex is assigned a degree, based on the degree distribution  $\{n_d\}$ . So we arbitrarily assign  $n_1$  vertices to have degree 1,  $n_2$  to have to degree 2, etc. We then partition the  $n$  vertices into affinity blocks. In general, an affinity block contains  $d + 1$  vertices that are assigned degree  $d$ , where  $d$  varies over the entire range. Note that there are many small blocks with vertices of low degree, and a few large blocks of high degree vertices. At this stage, no edges have been added.

*Phase 1.* This phase adds edges *within* each affinity block. Each block is a dense Erdős-Rényi graph, where the density depends on the size of the block. For a block involving degree  $d$  vertices, the density is chosen based on  $c_d$  (the observed clustering coefficient). All the parameters are chosen to ensure that each vertex achieves its desired clustering coefficient and is not incident to more edges than its desired degree.

*Phase 2.* This phase adds edges between the blocks. Consider some vertex  $i$  with an assigned degree  $d_i$ . Suppose it is already incident to  $d'_i$  edges from Phase 1. We set  $e_i = d_i - d'_i$  to be the *excess degree* of  $i$ . We must create  $e_i$  edges incident to vertex  $i$  to satisfy its degree requirement. We construct a Chung-Lu graph with degree sequence  $e_1, e_2, \dots, e_n$  to complete the graph construction.

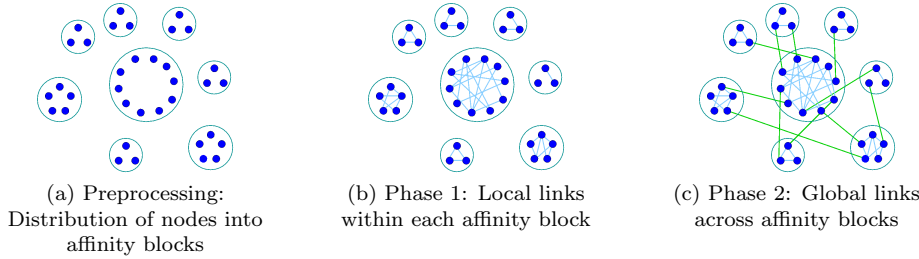


Fig. 2.1: BTER model phases [36].

**2.2. Developing a scalable implementation.** Our goal is to show that it is possible to have a highly scalable implementation of the BTER method. The main goal is to have *independent edge insertions* so that the edge generation can be parallelized.

As stated, Phase 2 edge insertions must happen after Phase 1, because we need to know the excess degrees. We parallelize this process by computing the *expected excess degree*. Given all the input parameters, we can precompute the expected excess degree for any vertex (this requires the compact representations and data structures) during the preprocessing. From this, we can precompute the total number of Phase 1 and Phase 2 edges.

To perform a parallel edge insertion, we first decide randomly whether this should be a Phase 1 or Phase 2 edge. For a Phase 1 edge, we select a random affinity block (with the appropriate probability) and connect two uniform members of the block.

For a Phase 2 edge, we perform a Chung-Lu edge insertion based on expected excess degrees. Because every edge is generated independently, there will be duplicates, but these are discarded in the final graph.

Given the structure of parallel edge insertion, the main challenges in developing a scalable implementation are as follows:

- **Preprocessing data structures.** A naïve implementation of the preprocessing step would require  $O(n)$  variables and storage. We design compact representations and data structures for the affinity blocks. This contains all the relevant information with minimal storage.
- **Repeats in Phase 1.** A direct parallel implementation of Phase 1 leads to many repeated edges, and this affects the overall degree distribution when edge repeats are removed. We model this problem as a *coupon collector problem* and give formulas for the number of extra Phase 1 edges to be inserted to rectify this.
- **The degree-1 problem of Phase 2.** A parallel implementation of Phase 2 results in numerous degree 1 vertices becoming isolated. We use a fix for this proposed in [14], which is different than the one proposed in the original BTER paper [36].

Once these issues are addressed, we have an embarrassingly parallel edge generation algorithm that requires only  $O(\log d_{\max})$  work per edge. The remainder of this section gives an in-depth but informal presentation of our implementation. A detailed algorithm specification is provided in the appendix.

**2.3. Preprocessing.** First, some notation. We let  $d_{\max}$  denote the maximum degree. We let  $d_i$  denote the (desired) degree of vertex  $i$ . For convenience, the nodes are indexed by increasing degree *except* for degree-1 nodes, which are indexed last. Hence, if  $d_i, d_j \geq 2$  and  $i < j$ , then  $d_i \leq d_j$ . As an example, see the numbering in Figure 2.2.

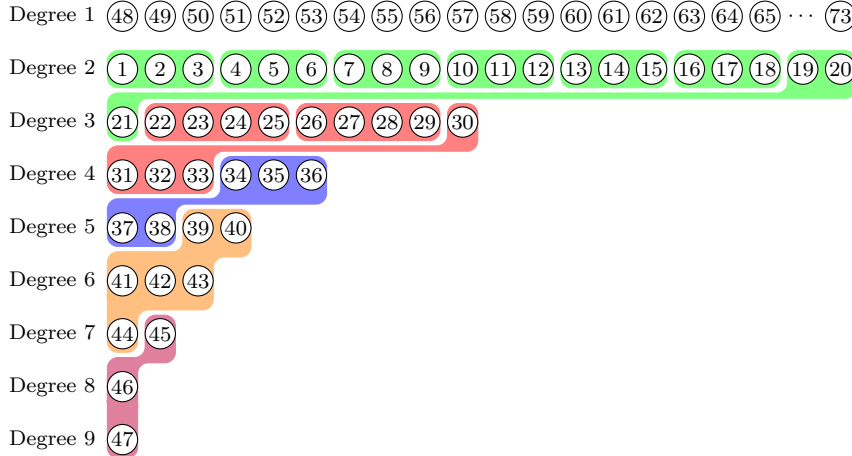


Fig. 2.2: Example of affinity blocks and groups.

In the preprocessing phase, we assign nodes to affinity blocks. For the assignment to affinity blocks, degree-1 nodes are ignored. The remaining nodes are assigned to affinity blocks in order (of degree). A *homogeneous* affinity block has  $d + 1$  nodes

of degree  $d$ . In Figure 2.2, the blocks are denoted by colored ovals, and 4-5-6 is a homogeneous block. The vast majority of (low-degree) nodes are assigned to homogeneous affinity blocks. However, there are not always enough nodes of degree  $d$  to fill in a homogeneous block; therefore, we also have a few (at most  $d_{\max}$ ) *heterogeneous* affinity blocks with nodes of different degrees. For instance, in Figure 2.2, 19-20-21 is a heterogeneous block.

Observe that storing a single block requires only three pieces of information: the starting index, the block size, and the block weight (related to its desired connectivity, which is a function of block size and clustering coefficient for the minimum degree in the block). However, note further that all affinity blocks of the same size and minimum degree can be grouped together into an *affinity group* — all blocks in the same group share the same block size and weight. In Figure 2.2, all nodes with the same color are in the same affinity group, e.g., 1-21 are in the same affinity group, likewise nodes 22-33, etc. The information needed to store an affinity group boils down to 4 items of information: the starting index of the group, the number of blocks in the group, the size of each block, and the weight of each block. The maximum number of groups is bounded by  $d_{\max}$ , so we need to store at most  $4 \cdot d_{\max}$  values.

Phase 2 needs to know the expected excess degree of all nodes, which is the difference between the desired degree and the number of expected links from Phase 1. Again, a naïve implementation would require  $O(n)$  information, but most nodes of the same degree behave the same. In a block where all nodes have the same degree, we say the nodes are *bulk nodes*. In a block with nodes of differing degrees, all nodes with degree equal to the minimum degree are still bulk nodes. We refer to the remaining nodes as *filler nodes*. In Figure 2.2, nodes 1-20 are degree-2 bulk nodes, nodes 22-30 are degree-3 bulk nodes, nodes 34-36 are degree-3 bulk nodes, and so on. Node 21 is a degree-3 fill node, nodes 31-33 are degree-4 filler nodes, etc. It is possible to have either no bulk nodes or no filler nodes for a given degree. In Figure 2.2, there are no filler degree-2 nodes and no bulk degree-6 nodes. Observe all bulk nodes of degree  $d$  (for any  $d$ ) are in blocks of the same size and connectivity; therefore, they all have the same excess degree. The filler nodes of degree  $d$  (for any  $d$ ) participate in at most one block and so all have the same excess degree. This means that there are two possible values for excess degree for the set of nodes with desired degree  $d$ . For each degree  $d$ , Phase 2 needs just 5 values: the number of filler and bulk nodes, the excess degree for filler nodes and for bulks nodes, and the starting index. (Technically, the starting index can be recomputed from  $n_d$ , but it reduces the work to store these indices explicitly. Likewise, we need not store both the bulk and filler counts so long as we keep the total number of nodes per degree.) Hence, the total working storage for Phase 2 information is  $5 \cdot d_{\max}$  values.

See the algorithm in Appendix A for details. The total storage (including inputs) needed by the generation routine is  $10 \cdot d_{\max}$  values. It is possible to modify the core data structures to store only the distinct degrees instead of maintaining a continuum of degrees until  $d_{\max}$ . This would change the store requirement to  $O(d_{\text{uniq}})$  instead of  $O(d_{\max})$ , where  $d_{\text{uniq}}$  is the number of distinct degrees in the graph. However, we present our ideas based  $O(d_{\max})$  storage for clarity of presentation.

**2.4. Phase 1.** Phase 1 creates intra-block links. Each affinity block is modeled as an Erdős–Rényi graph. An overwhelming majority of the triangles are formed in this phase, and thus we pick the Erdős–Rényi constant,  $\rho$ , for the block to match the target clustering coefficient  $c$ . A vertex of degree  $d$ , and clustering coefficient  $c$  is incident to  $c \cdot \binom{d}{2}$  triangles. Assume this vertex is grouped with other vertices of

degree  $d$  into block with  $d + 1$  vertices, which holds for the a great majority of the vertices. If we build an Erdős–Rényi graph of this block with parameter  $\rho$ , then this vertex is expected to be incident to  $\binom{d}{2}\rho^3$  triangles. Solving for  $\rho$  yields  $\rho = \sqrt[3]{c}$ . Therefore, for block  $b$ , the connectivity is  $\rho_b = \sqrt[3]{c_{d_b}}$  where  $d_b$  denotes the minimum degree in the block (since most blocks are homogeneous, this choice works well). Note that the clustering coefficients of vertices will be higher if we only consider the affinity blocks. This is to compensate for the edges that will be added in Phase 2 to increase the number of wedges, likely without contributing any triangles.

The difficulty in phase 1 is that we expect a preponderance of repeat edges in the case where edges are generated independently with replacement. Consider affinity block  $b$  with  $n_b$  nodes and connectivity  $\rho_b$ , meaning that each node in block  $b$  wants internal degree  $\rho_b \cdot d_b$ . BTER wants approximately  $m_b = \rho_b \binom{n_b}{2}$  *unique* edges in block  $b$ . Determining the number of draws with replacement to get a desired number of unique items can be cast as a *coupon collector problem*. We can show that a good approximation for the expected number of edges that need to be inserted is

$$w_b = \binom{n_b}{2} \ln(1/(1 - \rho_b)). \quad (2.1)$$

The proof is provided in [Appendix B](#).

We illustrate the utility of equation (2.1) with an example with  $n_b = 10$  nodes and connectivity  $\rho_b = 0.5$ , corresponding to  $m_b = 22.5$  edges, on average. In this case, the formula predicts that we need to do  $w_b = 31.1916$  draws, in expectation, to see the desired number of unique edges, in expectation. We do 10000 random experiments as follows. For  $i = 1, \dots, 10000$ , the random variable  $X_i \sim \text{Poisson}(w_b)$  is the number of items *drawn* from the  $\binom{n_b}{2} = 45$  possible edges, and  $Y_i$  is the number of those items that are *unique*. A histogram of the  $Y_i$  values is shown in [Figure 2.3](#). The average number of unique items exactly corresponds with the desired value.

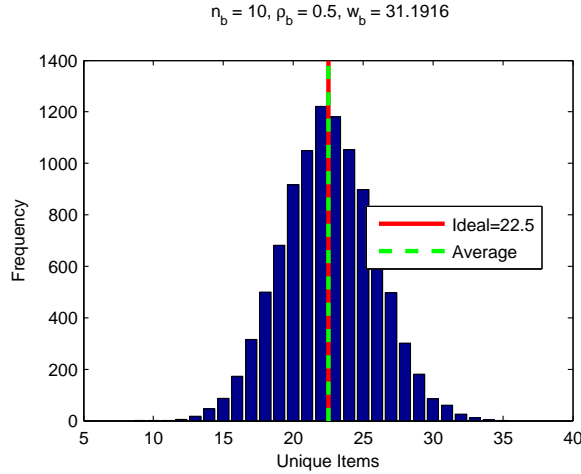


Fig. 2.3: Distribution of the number of unique edges on 10,000 random trials.

From equation (2.1), we can determine the number of extra edges needed in Phase 1. Specifically, we insert  $w^{(1)} = \sum_b w_b$  edges to get a total of  $m^{(1)} = \sum_b m_b$  unique



Phase 1 edges. Given that we are generating a Phase 1 edge, the process proceeds as follows:

1. Pick an affinity group randomly proportional to the total weight of its constituent blocks.
2. Pick a block within the affinity group uniformly at random (all blocks within the same group have the same weight).
3. Pick two nodes *without replacement* uniformly at random from the selected block — these two nodes form an edge.

The first step is a weighted sampling step and requires  $O(\log g_{\max})$  work, where  $g_{\max} \leq d_{\max}$  is the total number of affinity groups. The other 2 steps are constant time operations.

**2.5. Phase 2.** Phase 2 is simply a Chung-Lu model on the expected excess degrees. In creating an edge, we choose two nodes independently. Those nodes are chosen proportional to their excess degree. For node  $i$  in group  $b$ , let  $w_i = \frac{1}{2}[d_i - (\rho_b \cdot d_b)]$  denote half its excess degree. The total number of edges that should be inserted in Phase 2 is  $w^{(2)} = \sum_i w_i$ . Duplicate edges are fairly rare, so we expect  $m^{(2)} \approx w^{(2)}$  Phase 2 edges.

Let  $n_d^{\text{fill}}$  and  $n_d^{\text{bulk}}$  be the number of filler and bulk nodes of degree  $d$ , let  $w_d = \sum_{i \in V_d} w_i$  be the weight of all degree- $d$  nodes, and let  $r_d$  be the proportion that are filler nodes. Inserting a Phase 2 edge proceeds as follows:

1. Pick degree  $d$  proportional to  $w_d$ .
2. Pick between filler and bulk nodes, according to  $r_d$ .
3. Pick a filler or bulk node (depending on the outcome of Step 2) of degree  $d$ , uniformly at random.

The first step is a weighted sampling step and required  $O(\log(d_{\max}))$  work, while the other two steps are constant time operations.

One complication in Phase 2 is that getting the correct number of degree-1 nodes poses a problem — approximately 36% of the pool of potential degree-1 nodes will not be selected and another 28% will have degree 2 or larger. A fix for this problem has been proposed in [14], which involves increasing the pool of degree-1 nodes, without changing the the expected number of edges that will be connected to these vertices. This increase in the pool size is controlled by the “blowup factor,”  $\beta \geq 1$ . This is included in the algorithm described in [Appendix A](#).

**2.6. Independent Edge Generation.** Lastly, we pull everything together to explain the independent edge generation. We insert a total of  $w = w^{(1)} + w^{(2)}$  edges, flipping a weighted coin for each edge to determine if it is Phase 1 or Phase 2. We expect to generate a total of  $m = m^{(1)} + m^{(2)}$  edges.

Generating the edges is extremely inexpensive:  $O(\log(d_{\max}))$  per edge. The expensive step is de-duplication where extra copies of repeated edges are removed. The same difficulty exists for the current Graph 500 (SKG) benchmark. Some may argue that duplicate edges are a useful feature since real data also has duplicates, but it is not clear that the duplication rates are similar to those observed in real data.

**2.7. Implementations.** We provide the algorithm in [Appendix A](#). We have a reference implementation in MATLAB that will be made available in the future. We have also implemented the method in Hadoop and use this in some of our experiments. The implementation is straightforward — we divide evenly the work of creating the  $w$  edges between a user-specified number of mappers. Each edge is hashed, and that



hash is the key for the reduce phase. The reducers remove any duplicate edges and emit a final list of all edges.

### 3. Numerical Comparisons.

**3.1. Small data.** In Figure 3.1, we present comparisons of BTER with the state-of-the-art in scalable generative models: SKG (current Graph 500 generator) [8, 23, 42] and CL [1, 10, 11, 14] on two small data sets available from SNAP [44]. We treat all edges as undirected and remove any duplicate edges and loops. The graph ca-AstroPh is a collaboration network based on 124 months of data from the astrophysics section of the arXiv preprint server; it has 9,987 nodes and 25,973 edges. The graph soc-Epinions1 is a who-trusts-whom online social (review) network from the Epinions website with 75,879 nodes and 405,740 edges.

The parameters of SKG are from [23], obtained from their KronFit algorithm. The input to CL is the degree distribution of the real graph and a blowup factor of 10 (same as is used for BTER for better matching degree-1 vertices [14]). The inputs to BTER are the degree distribution and clustering coefficients per degree (computed exactly) of the real graph and a blowup factor of 10. Timings are not reported as they are negligible for all three methods.

*Degree distribution.* The degree distributions for the original graphs and the models are shown in Figures 3.1a and 3.1d. SKG is known to have oscillations in the degree distribution [39, 37], and these oscillations are easily visible in Figure 3.1d. The oscillations are correctable with appropriate addition of noise [39, 37] (not shown), but even then it tends to over estimate the low degree nodes and miss the highest degree nodes. In contrast, both CL and BTER closely match the real data.

*Clustering coefficients.* The clustering coefficients per degree for the original graphs and the models are shown in Figures 3.1a and 3.1e. The SKG graph model has no inherent mechanism for closing triangles and creating community structure. Though a few triangles may close at random, they are insufficient for the SKG-generated graph to match the clustering coefficients in the real data. The situation for CL is similar to that for SKG — there is no reason for wedges to close. BTER, on the other hand, provides a much closer match to the real data.

*Eigenvalues of adjacency matrix.* We show the top 50 leading eigenvalues (in magnitude) of the adjacency matrix in Figures 3.1c and 3.1f. BTER is a much closer match to the real data — especially the first few eigenvalues. Under certain circumstances, matching the degree distribution should produce a match in eigenvalues [25]. However, we conjecture that graphs with community structure require that triangle structure also be matched to obtain a good fit for the eigenvalues.

**3.2. Large data.** We demonstrate that BTER is able to fit large-scale real-world data. We do not compare to SKG because it is not possible to fit the parameters for such large graphs. We do not compare to the CL model because we can easily explain the performance: its match in terms of the degree distribution is nearly identical to that of BTER, and its clustering coefficients are close to zero, as for the small data. The data sets are described in Table 3.1a. We treat all edges as undirected and remove any duplicate edges and loops. We obtained real-world graphs from the Laboratory for Web Algorithms [43], which compressed the graphs using LLP and WebGraph [7, 5]. Briefly, the networks are described as follows.

- amazon-2008 [7, 5]: A graph describing similarity among books as reported by the Amazon store.

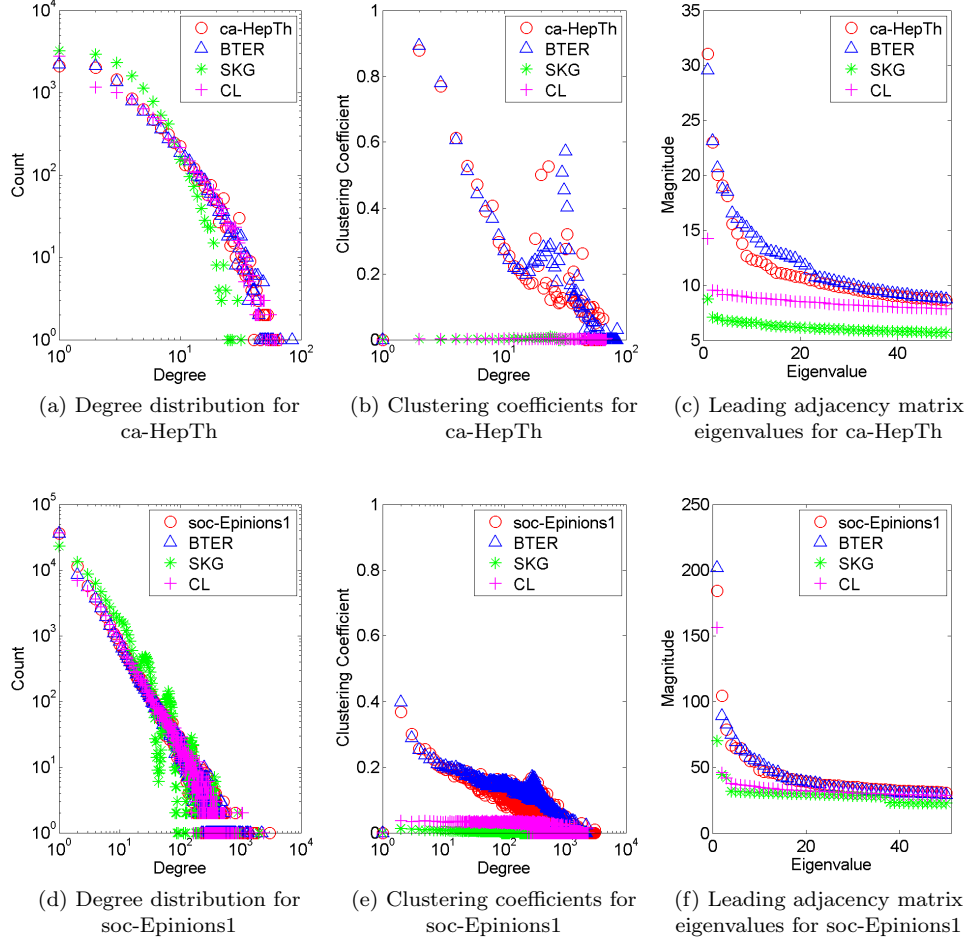


Fig. 3.1: Comparison of CL, SKG, and BTER on small graphs.

- *ljournal-2008* [9, 7, 5]: Nodes represent users on LiveJournal. Node  $x$  connects to node  $y$  if  $x$  registered  $y$  as a friend.
- *hollywood-2011* [7, 5]: This is a graph of actors. Two actors are joined by an edge whenever they appear in a movie together.
- *twitter-2010* [22, 7, 5]: Nodes are Twitter users, and node  $x$  links to node  $y$  if  $y$  follows  $x$ .
- *uk-union-2006-06-2007-05* (shorted to *uk-union*) [6, 7, 5]: Links between web pages on the .uk domain. We ignore the time labeling on the links.

The smaller graphs (*amazon-2008*, *ljournal-2008*, *hollywood-2011*) are those with up to roughly 100M edges. These can be easily processed using MATLAB on an SGI Altix UV 10 with 32 cores (4 Xeon 8-core 2.0GHz processors) and 512 GB DDR3 memory. None of the parallel capabilities of MATLAB are enabled for these studies. To give a sense of the memory requirements, storing the *hollywood-2011* graph as a sparse matrix in MATLAB requires 3.4GB of storage. The larger graphs (*twitter-*

Table 3.1: Network characteristics of original and BTER-generated graphs.

(a) Large data set properties.

Graph	$ V $	$ E $	$d_{\max}$	$d_{\text{avg}}$	GCC
amazon-2008	1M	4M	1,077	10	0.260
ljournal-2008	5M	50M	19,432	18	0.124
hollywood-2011	2M	114M	13,107	115	0.175
twitter-2010	40M	1,202M	2,997,487	60	0.001
uk-union	122M	4,659M	6,366,528	76	0.007

(b) Properties of BTER-generated graphs, including generation and edge deduplication time.

Graph	$ V $	$ E $	$d_{\max}$	$d_{\text{avg}}$	GCC	Gen.	Dedup.
amazon-2008	1M	4M	1,052	10	0.253	2.27s	9.25s
ljournal-2008	5M	49M	18,510	19	0.127	33.81s	126.40s
hollywood-2011	2M	114M	11,676	115	0.180	88.54s	362.25s
twitter-2010	38M	1,133M	1,635,823	59	0.004	222.87s	
uk-union	120M	4,399M	1,497,950	73	0.111	1638.28s	

2010, uk-union) each have over 1B edges. These are processed on a Hadoop cluster with 32 compute nodes. Each compute node has an Intel i7 930 CPU at 2.8GHz (4 physical cores, HyperThreading enabled), 12 GB of memory, and 4 2TB SATA disks. All experiments were run using Apache Hadoop version 0.20.203.0<sup>1</sup>.

The inputs to BTER are the degree distribution and clustering coefficients by degree. (We used a blowup of  $\beta = 1$  for the experiments reported here.) Computing the degree distribution is straightforward. However, for the clustering coefficients calculations, we used the sampling approach as implemented in [20] with 2000 samples per degree, so the expected error is  $\epsilon = 0.05$  at a confidence level of 99.9%. Sampling was not required for the smaller graphs, but we have used it in all experiments for consistency.

*BTER Timing.* Table 3.1b shows the details and timings for the graphs produced by BTER. Observe the close match in the characteristics of the graphs in terms of number of nodes, number of edges, maximum degree, average degree, and global clustering coefficient. For the smaller graphs, we are able to separate the edge generation and deduplication time. The generation time is not strictly proportional to the number of desired edges because we have to generate extra edges for Phase 1 to account for possible duplicates (see §2.4). The parallelism of Hadoop yields a large advantage in terms of time. The twitter-2010 graph has 10 times more edges than hollywood-2011, but it takes less than half the time to do the computation on the 32-node Hadoop cluster.

*Degree Distribution.* Figure 3.2 illustrates the match between the real data and the BTER graph. BTER cannot easily match discontinuities in the degree distribution because of the randomness in creating edges. The issue is that nodes generally do not get *exactly* the desired degree—it may be one or two more or less. For a smooth degree distribution, neighboring degrees cancel the effect on one another. For discontinuous distributions, the BTER degree distribution is a “smoothed” version. This is evident, for instance, in the amazon-2008 data where we can see a smoothing effect on the

<sup>1</sup><http://hadoop.apache.org/>

sharp discontinuity near degree 10.

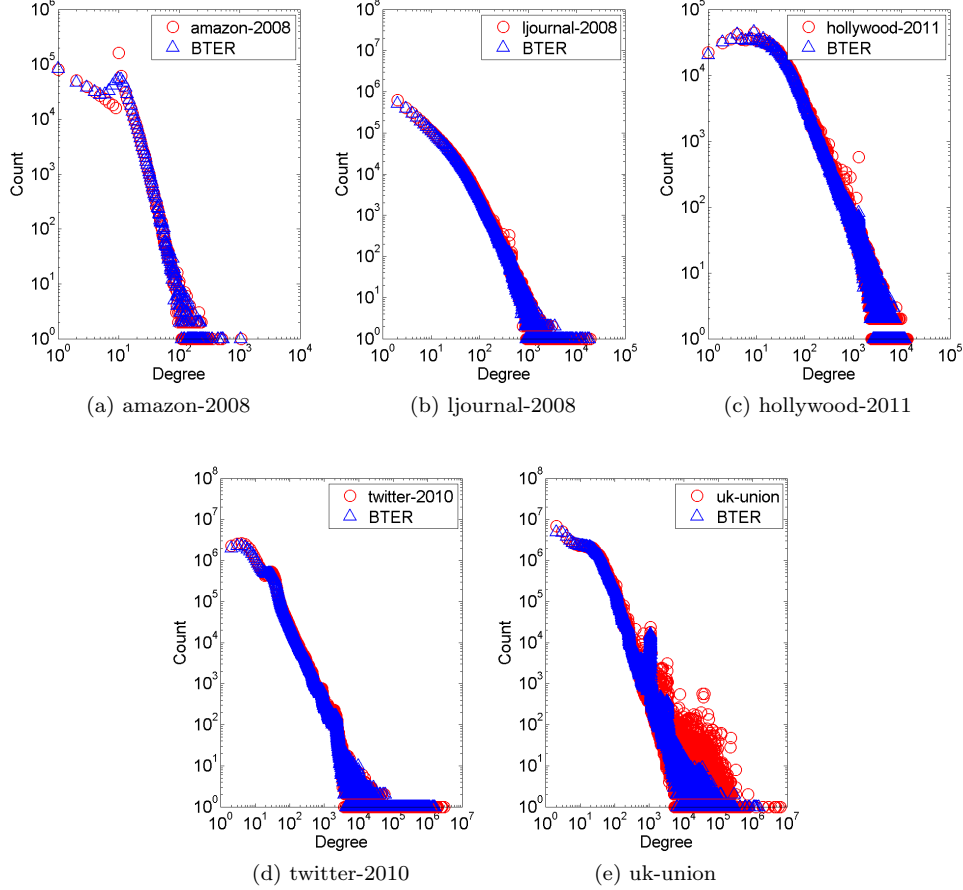


Fig. 3.2: Degree distributions of original and BTER-generated graphs.

*Clustering Coefficients.* BTER’s strength is its ability to match clustering coefficients and therefore community structure. Most degree distributions are heavy tailed and have a relatively consistent structure. The same is not true for clustering coefficients. Different profiles can potentially lead to graphs with fundamentally different structures. Figure 3.3 shows the clustering coefficients of the real data and the BTER-generated graphs. There is a very close match.

**4. Proposed Benchmark.** So far we have shown that BTER can be used to match real-world data. For benchmarking purposes however, reasonable “ideal” profiles for degree distribution and clustering coefficient by degree are required. In this section, we propose some possibilities, noting that these can easily be changed for whatever scenario a user may encounter.

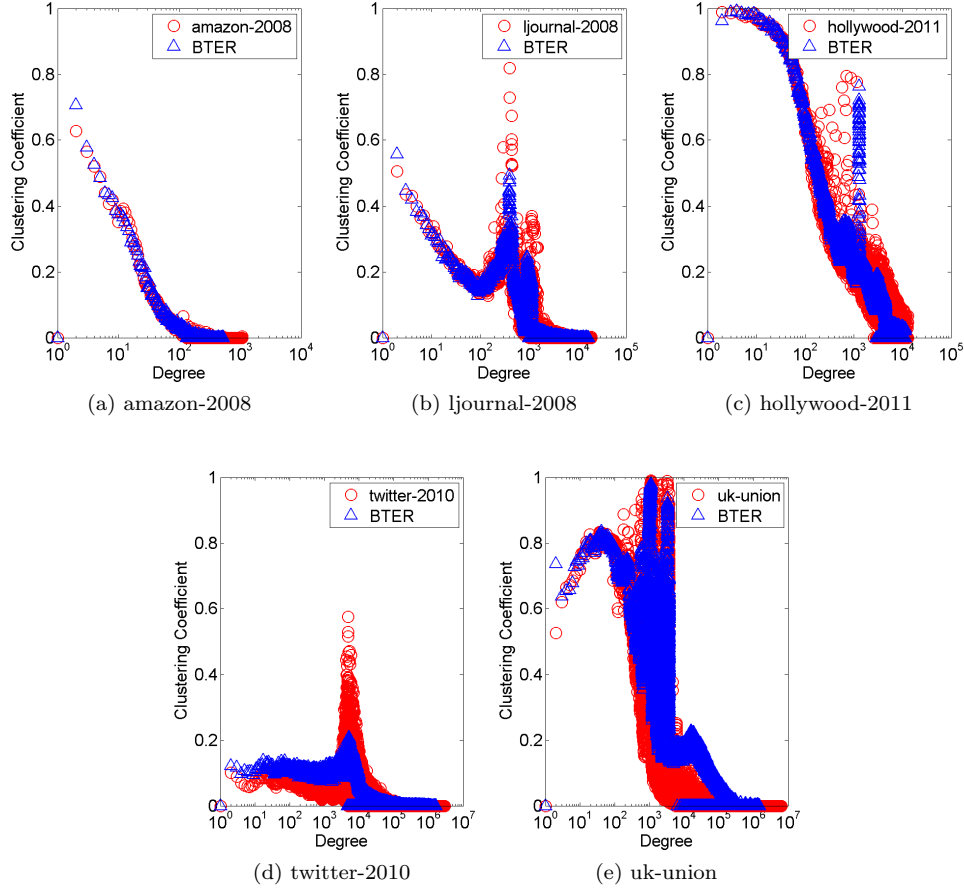


Fig. 3.3: Clustering coefficients of original and BTER-generated graphs.

**4.1. Idealized Degree Distribution.** It has been hypothesized that degree distribution of real-world networks follow a power law (PL) degree distribution, i.e.,

$$n_d \propto d^{-\gamma},$$

for some parameter  $\gamma$  [2]. However, our observation is that power law distributions are difficult to use as a model — a point that is discussed in more detail below. It has been suggested that power laws are not necessarily the best descriptors for real-world networks [35, 4]. Finally, proving (in a statistical sense) that a single observed degree distribution is power law is difficult [12].

For benchmarking purposes, our goal is to specify an ideal average degree,  $\bar{d}$ , and an absolute bound on maximum degree,  $d^*$ . Let  $f(d)$  define the desired proportionality of degree  $d$ . We then create a *discrete* distribution on  $d = 1, \dots, d^*$  as

$$\Pr(D = d) = \frac{f(d)}{\sum_{d'=1}^{d^*} f(d')}.$$

Ideally, the average degree is equal to  $\bar{d}$  and the probability of having degree  $d^*$  is sufficiently small, i.e.,

$$\bar{d} = \sum_{d=1}^{d^*} d \cdot f(d) \quad \text{and} \quad \Pr(D = d^*) < \epsilon_{\text{tol}},$$

where  $\epsilon_{\text{tol}}$  is small enough such that  $n \cdot \epsilon_{\text{tol}} \ll 1$  (where  $n$  is the number of nodes). For the power law distribution, it can be difficult to find a value for  $\gamma$  that yields a high enough average degree and a low enough probability of choosing  $d^*$ . Hence, we propose instead a generalized log-normal (GLN) distribution, i.e.,

$$n_d \propto \exp \left[ - \left( \frac{\log x}{\alpha} \right)^\beta \right],$$

for some parameters  $\alpha$  and  $\beta$ . Additionally, the shape of the distribution is typical of the real-world graphs shown in §3.

We consider two scenarios, both with  $n = 10^7$  nodes. We do a parameter search on  $\alpha$  and  $\beta$  (`fminsearch` in MATLAB) to locate the optimal parameters. A function for finding the optimal parameters for either discrete GLN or discrete PL for user-specified values of  $d_{\text{avg}}$  and  $d_{\text{max}}$  is included in the reference code to be released at a future date.

*Scenario 1 for Degree Distribution Fitting.* In the first scenario, the targets are  $\bar{d} = 16$  and  $d^* = 10^6$ . For discrete PL, the optimal parameter is  $\gamma = 1.911$  with  $d_{\text{avg}} = 16$  and  $\Pr(D = d^*) = 1.97 \times 10^{-12}$ . For discrete GLN, the optimal parameters are  $\alpha = 1.988$  and  $\beta = 2.079$  with  $d_{\text{avg}} = 16$  and  $\Pr(D = d^*) = 4.14 \times 10^{-26}$ . Realizations of the two distributions are pictured in Figure 4.1a. For this scenario, both degree distributions are reasonable in that there is no sharp drop off as we get close to the maximum allowable degree,  $d^*$ .

*Scenario 2 for Degree Distribution Fitting.* In the second scenario, the targets are  $\bar{d} = 64$  and  $d^* = 10^5$ . For PL, the optimal parameter is  $\gamma = 1.668$  with  $d_{\text{avg}} = 64$  but  $\Pr(D = d^*) = 2.16 \times 10^{-9}$  (fairly large). For discrete GLN, the optimal parameters are  $\alpha = 2.171$  and  $\beta = 1.877$  with  $d_{\text{avg}} = 64$  and  $\Pr(D = d^*) = 8.35 \times 10^{-12}$ . Realizations of the two distributions are pictures in Figure 4.1b. In this scenario, the problem with power law becomes apparent — near  $d^*$ , there are still many degrees with *multiple* nodes so that the cutoff is extremely abrupt. In comparison, discrete GLN fades more naturally to the desired maximum degree.

**4.2. Idealized Clustering Coefficients.** As there is no definitive structure to clustering coefficients, we propose a simple parameterized curve that has some similarity to real data observations.

Let  $\{n_d\}$  define the specified degree distribution and  $d_{\text{max}}$  be the maximum degree such that  $n_d > 0$ . We define  $\bar{c}_d$ , the mean value for  $c_d$ , as

$$\bar{c}_d = c_{\text{max}} \exp(-(d-1)^\xi) \quad \text{for } d \geq 2,$$

where  $c_{\text{max}}$  and  $\xi$  are parameters. If  $c_{\text{max}}$  is specified, then a simple parameter search can be used to fit  $\xi$  to a target global clustering coefficient; code to fit the data is included in the reference code. The final values are for  $\{c_d\}$  are selected as

$$c_d \sim \mathcal{N}(\bar{c}_d, \min\{10^{-2}, \bar{c}_d/2\}).$$

The randomness could, of course, be omitted.

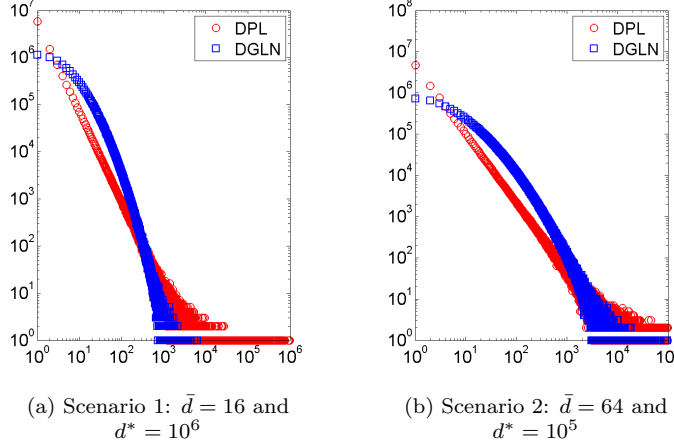


Fig. 4.1: Example degree distributions from discrete power law (DPL) and discrete generalized log normal (DGLN) for  $n = 10^7$  nodes.

**4.3. Example Graphs.** We generate two example graphs per the scenarios below. Table 4.1 lists the network characteristics and Figure 4.2 shows the target and BTER-generated degree distributions and clustering coefficients.

Table 4.1: Network characteristics of BTER-generated graphs for benchmarking.

Graph	$ V $	$ E $	$d_{\max}$	$d_{\text{avg}}$	GCC	Gen.	Dedup.
Ideal 1	1M	35M	28,643	72	0.406	35.11s	117.18s
Ideal 2	1M	8M	2,594	17	0.104	5.07s	20.66s

*Scenario 1.* For the first set-up, we selected  $\bar{d} = 75$  and  $d^* = 100,000$  to define the degree distribution. The parameter search selected  $\alpha = 2.14$  and  $\beta = 1.83$ . For the clustering coefficients, we set  $c_{\max} = 0.9$  and targets a GCC of 0.15. The parameter search selected  $\xi = 3.59 \times 10^{-4}$  for defining the clustering coefficient profile.

*Scenario 2.* For the second set-up, we selected  $\bar{d} = 16$  and  $d^* = 10,000$  to define the degree distribution. The parameter search selected  $\alpha = 1.98$  and  $\beta = 2.08$ . For the clustering coefficients, we set  $c_{\max} = 0.5$  and targets a GCC of 0.10. The parameter search selected  $\xi = 0.01$  for defining the clustering coefficient profile.

**5. Conclusions.** This paper demonstrates that the BTER generative model is appropriate for modeling massive networks. We provide a detailed algorithm along with analysis explaining the workings of the method. The original paper on BTER [36] provided none of the implementation details and, in fact, did not directly use the clustering coefficient data but rather estimated it via a function. Here we give precise details on the implementations, which is nontrivial due to issues such as repeat edges. We are able to build a model of a graph with 120M nodes and 4.7M edges in less than 30 minutes on a 32-node Hadoop cluster.

The development of a realistic graph model is an important step in developing effective “null” models that nonetheless share the properties of real-world networks.



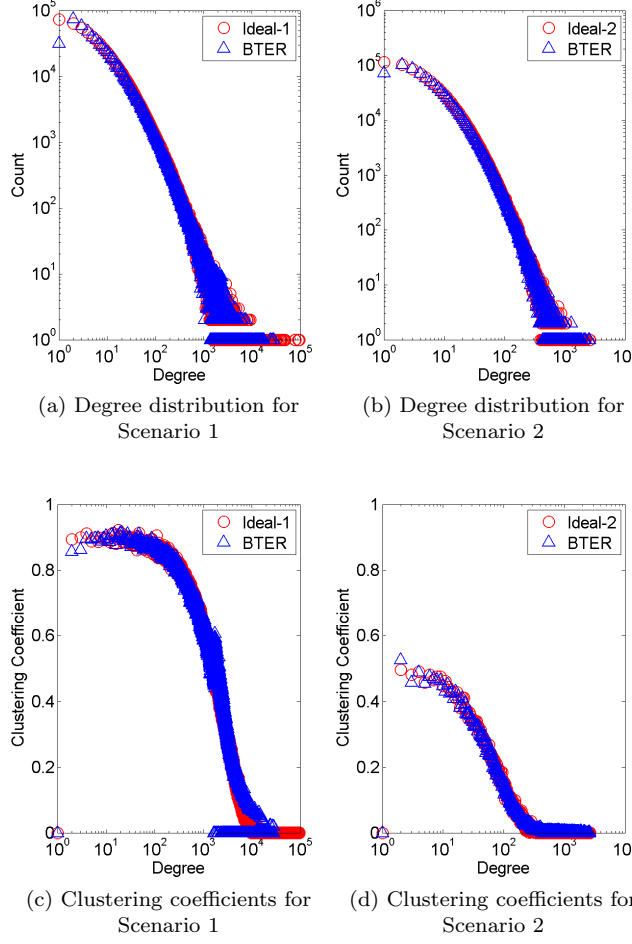


Fig. 4.2: Target distributions and results of BTER-generated graphs.

Such models will be useful in detecting anomalies, statistical sampling, and community detection. For example, the BTER model does not have larger communities beyond the affinity blocks, whereas we might expect that real-world graphs have a richer structure such as a hierarchy or other complex behavior.

We believe the proposed BTER model, along with the proposed degree and clustering coefficient distributions, will boost benchmarking efforts in graph processing by providing realistic graph instances. The proposed degree distributions capture the essence of degree distributions that we see in practice and generate realistic distributions even at large scales (whereas power law has a reputation of generating a few degrees that are much larger than observed in practice). Moreover, the proposed distribution allows us to modify both the average and the maximum degree, which is critical for benchmarking. The proposed clustering coefficient curves implicitly embed triangle structure into the graphs, which is a critical feature that distinguishes real graphs from arbitrary sparse graphs. And finally, the proposed generation algo-

rithm enables generating extremely large graphs thanks to its parallel edge generation property. All together, we believe the proposed model will fill a much needed hole in benchmarking graph processing

Of course, we only consider the case of a static, unattributed, undirected network. Future work will be aimed at developing models that capture dynamics, attributes, and direction, but even capturing the degree distribution in a directed graph poses challenges [14].

## Appendix A. BTER algorithm details.

For convenience, notation is described in Table A.1. The node- and block-level variables are not used in the algorithms.

**A.1. Preprocessing.** The BTER Setup procedure is described in Algorithm 1. The inputs are the degree distribution,  $\{n_d\}$ ; the clustering coefficients per degree,  $\{c_d\}$ ; and the blowup factor for degree-1 nodes,  $\beta$ .

The method precomputes the index for the first node of each degree,  $\{i_d\}$ , and the number of nodes with degree greater than the degree  $d$ ,  $\{n'_d\}$ . The latter is not saved.

The degree-1 nodes are handled specially. All degree-1 nodes are arbitrarily assigned to be “fill” nodes. The number of candidate degree-1 nodes may be increased using the blowup factor,  $\beta$ . However, if the blowup is used, the majority of the (desired) degree-1 nodes will ultimately have degree 0 and can be removed in post-processing.

The main loop walks through each degree, determining the information for Phases 1 and 2.

It first allocates degree- $d$  nodes to be fill nodes for the last incomplete block, if needed. The number of nodes necessary to complete the last incomplete block is denoted by  $n_*^{\text{fill}}$ . The excess degree of any fill nodes depends on the internal degree of the last incomplete block, denoted by  $d_*$ . The excess degree is used to determine the weight of the degree- $d$  fill nodes for phase 2,  $w_d^{\text{fill}}$ .

If more nodes of degree  $d$  remain, these are allocated as bulk nodes and a new group is formed. The number of bulk nodes of degree  $d$  is denoted by  $n_d^{\text{bulk}}$ . For the new group, we determine the index of the first node, the number of blocks, and the size of each block. The very last block of the very last group is special because the remaining nodes may not be enough to fill it. For simplicity and because it is often the case for heavy-tailed networks, we assume the last group contains only a one block. This makes handling it as a special case simpler. We compute excess degree for these bulk nodes and then the corresponding weight of degree- $d$  bulk nodes for phase 2,  $w_d^{\text{bulk}}$ . We also compute the weight of the group,  $w_g$ , using the coupon collector over-sampling weight described in §2.4. Finally, we compute the number of nodes needed to fill out the last block,  $n_*^{\text{fill}}$ .

Rather than returning  $w_d^{\text{fill}}$  and  $w_d^{\text{bulk}}$  directly, it is easier (for the edge generation phase) to have their sum,  $w_d$ , and the ratio of fill nodes,  $r_d$ . Likewise, we do not return  $n_d^{\text{bulk}}$  since it can be easily recomputed using  $n_d$  and  $n_d^{\text{fill}}$ . We do return  $i_d$ , but this could be omitted and recomputed if that were more efficient (e.g., reducing communication to workers). Finally, we no longer need to keep  $\{c_d\}$  after the preprocessing is complete.

**A.2. Generating Edges.** BTER edge generation is shown in Algorithm 2. The procedure RANDOM\_SAMPLE does a weighted sampling according to a specified discrete distribution. For  $p$  bins, the cost is  $O(\log(p))$ . For each edge, we randomly

---

**Algorithm 1** BTER Setup
 

---

```

1: procedure BTER_SETUP( $\{n_d\}, \{c_d\}, \beta$ )

    Number nodes from least degree to greatest, except degree-1 nodes are last
2:    $i_2 \leftarrow 1$ 
3:   for  $d = 3, \dots, d_{\max}$  do
4:      $i_d \leftarrow i_{d-1} + n_{d-1}$ 
5:   end for
6:    $i_1 \leftarrow i_{d_{\max}} + n_{d_{\max}}$ 

    Compute number of nodes with degree greater than  $d$ 
7:   for  $d = 1, \dots, d_{\max}$  do
8:      $n'_d \leftarrow \sum_{d' > d} n'_{d'}$ 
9:   end for

    Handle degree-1 nodes
10:   $n_1^{\text{fill}} \leftarrow \beta \cdot n_1, w_1 \leftarrow \frac{1}{2}n_1, r_1^{\text{fill}} \leftarrow 1$ 

    Main loop
11:   $g \leftarrow 0, n_*^{\text{fill}} \leftarrow 0, d_* \leftarrow 0$ 
12:  for  $d = 2, \dots, d_{\max}$  do
13:    if  $n_*^{\text{fill}} > 0$  then  $\triangleright$  Try to fill incomplete block from current group
14:       $n_d^{\text{fill}} \leftarrow \min(n_*^{\text{fill}}, n_d)$ 
15:       $n_*^{\text{fill}} \leftarrow n_*^{\text{fill}} - n_d^{\text{fill}}$ 
16:       $w_d^{\text{fill}} \leftarrow \frac{1}{2}n_d^{\text{fill}}(d - d_*)$ 
17:    else
18:       $n_d^{\text{fill}} \leftarrow 0, w_d^{\text{fill}} \leftarrow 0$ 
19:    end if
20:     $n_d^{\text{bulk}} \leftarrow n_d - n_d^{\text{fill}}$ 
21:    if  $n_d^{\text{bulk}} > 0$  then  $\triangleright$  Create a new group for degree- $d$  bulk nodes
22:       $g \leftarrow g + 1$ 
23:       $i_g \leftarrow i_d + n_d^{\text{fill}}$ 
24:       $b_g \leftarrow \lceil n_d^{\text{bulk}} / (d + 1) \rceil$ 
25:       $n_g \leftarrow d + 1$ 
26:      if  $b_g \cdot (d + 1) > (n'_d + n_d^{\text{bulk}})$  then  $\triangleright$  Special handing of last group
27:        if  $b_g \neq 1$  then throw error
28:         $n_g \leftarrow (n'_d + n_d^{\text{bulk}})$ 
29:      end if
30:       $\rho_* \leftarrow \sqrt[3]{c_d}$ 
31:       $d_* \leftarrow (n_g - 1) \cdot \rho_*$ 
32:       $w_d^{\text{bulk}} \leftarrow \frac{1}{2}n_d^{\text{bulk}} \cdot (d - d_*)$ 
33:       $w_g \leftarrow b_g \cdot \frac{1}{2}n_g(n_g - 1) \cdot \log(1/1 - \rho_*)$ 
34:       $n_*^{\text{fill}} \leftarrow (b_g \cdot n_g) - n_d^{\text{bulk}}$ 
35:    else
36:       $w_d^{\text{bulk}} \leftarrow 0$ 
37:    end if
38:     $w_d \leftarrow w_d^{\text{fill}} + w_d^{\text{bulk}}, r_d \leftarrow w_d^{\text{fill}} / w_d$ 
39:  end for

40:  return  $\{i_d\}, \{w_d\}, \{r_d\}, \{n_d^{\text{fill}}\}, \{w_g\}, \{i_g\}, \{b_g\}, \{n_g\}$ 

```

---

Table A.1: Description of variables. A ★ symbol in the second column indicates that this variable is explicitly computed and stored for use by the sampling procedure; a × symbol means the variable is not explicitly computed. The last column gives the formula to derive it from the stored values.

<i>Scalars</i>			
$n$		Total number of nodes	$n = \sum n_d$
$m$		Total number of edges	$m = \frac{1}{2} \sum d \cdot n_d$
$w$		Total number of edges to insert	$w = w^{(1)} + w^{(2)}$
$d_{\max}$	★	Largest (desired) degree	
$b_{\max}$	×	Total number of affinity blocks	$b_{\max} = \sum_g b_g$
$g_{\max}$	★	Total number of affinity groups	$g_{\max} \leq d_{\max}$
$\beta$	★	Blowup factor for degree-1 vertices	
<i>Node Level</i> $i = 1, \dots, n$			
$d_i$	×	Degree (desired) of node $i$	
$b_i$	×	Block id for degree $i$	
$w_i$	×	Excess degree of nodes $i$	$w_i = \frac{1}{2} [d_i - (\rho_{b_i} \cdot d_{b_i})]$
<i>Block Level</i> $b = 1, \dots, b_{\max}$			
$d_b$	×	Minimum degree in block $b$	
$\rho_b$	×	Connectivity of degree $b$	$\rho_b = \sqrt[3]{c_{d_b}}$
$n_b$	×	Number of nodes in block $b$	$n_b = d_b + 1$
$m_b$	×	Number of unique edges in block $b$	$m_b = \rho_b \binom{n_b}{2}$
$w_b$	×	Weight of block $b$	$w_b = \binom{n_b}{2} \ln(1/(1 - \rho_b))$
<i>Degree Level</i> $d = 1, \dots, d_{\max}$			
$n_d$	★	Number of nodes of degree $d$	
$c_d$	★	Mean clustering coefficient for nodes of degree $d$	
$i_d$	★	Index of first degree of degree $d$	
$n'_d$		Number of nodes of degree greater than $d$	$n'_d = \sum_{d' > d} n'_{d'}$
$n_d^{\text{fill}}$	★	Number of fill nodes of degree $d$	
$n_d^{\text{bulk}}$		Number of bulk nodes of degree $d$	$n_d^{\text{bulk}} = n_d - n_d^{\text{fill}}$
$w_d$	★	Excess degree of nodes of degree $d$	
$r_d$	★	Ratio of fill excess degree for degree $d$	
$w_d^{\text{fill}}$		Excess degree of fill nodes of degree $d$	$w_d^{\text{fill}} = r_d \cdot w_d$
$w_d^{\text{bulk}}$		Excess degree of bulk nodes of degree $d$	$w_d^{\text{bulk}} = w_d - w_d^{\text{fill}}$
<i>Group Level</i> $g = 1, \dots, g_{\max}$			
$i_g$	★	Index of first node in group $g$	
$b_g$	★	Number of affinity blocks in group $g$	
$n_g$	★	Number of nodes per block in group $g$	
$w_g$	★	Weight of group $g$ (including duplicate edges)	
<i>Phase Level</i> $k = 1, 2$			
$w^{(k)}$		Weight of phase $k$	$w^{(1)} = \sum_g w_g$ $w^{(2)} = \sum_d w_d$

select between the phases using a weighted coin. A Phase 1 edge requires one sample from a discrete distribution of size  $g_{\max}$  and three additional random values drawn uniformly from  $[0, 1]$ . A Phase 2 edges requires two samples from a discrete distribution of size  $d_{\max}$  and four additional random values drawn uniformly from  $[0, 1]$ . Since  $g_{\max} \leq d_{\max}$ , an upper bound on the cost per edge is the cost of one discrete random sample on a distribution of size  $d_{\max}$  plus four random values drawn uniformly from  $[0, 1]$ .

---

**Algorithm 2** BTER Sample

---

```
1: procedure BTER_SAMPLE( $\{n_d\}, \{i_d\}, \{w_d\}, \{r_d\}, \{n_d^{\text{fill}}\}, \{w_g\}, \{i_g\}, \{b_g\}, \{n_g\}$ )
2:    $w^{(1)} \leftarrow \sum_g w_g, w^{(2)} \leftarrow \sum_d w_d, w \leftarrow w^{(1)} + w^{(2)}$ 
3:    $E^{(1)} \leftarrow \emptyset, E^{(2)} \leftarrow \emptyset$ 
4:   for  $j = 1, \dots, w$  do
5:      $r \sim U[0, 1]$ 
6:     if  $r < w^{(1)}/w$  then
7:        $E^{(1)} \leftarrow E^{(1)} \cup \text{BTER\_SAMPLE\_PHASE1}(\{w_g\}, \{i_g\}, \{b_g\}, \{n_g\})$ 
8:     else
9:        $E^{(1)} \leftarrow E^{(1)} \cup \text{BTER\_SAMPLE\_PHASE2}(\{w_d\}, \{r_d\}, \{n_d\}, \{n_d^{\text{fill}}\}, \{i_d\})$ 
10:    end if
11:  end for
12:  return  $E^{(1)}, E^{(2)}$ 

13: procedure BTER_SAMPLE_PHASE1( $\{w_g\}, \{i_g\}, \{b_g\}, \{n_g\}$ )
14:    $g \leftarrow \text{RANDOM\_SAMPLE}(\{w_g\})$  ▷ Choose group
15:    $r_1 \sim U[0, 1], \delta = i_g + \lfloor r_1 \cdot b_g \rfloor \cdot n_g$  ▷ Choose block and compute its offset
16:    $r_2 \sim U[0, 1], i \leftarrow \lfloor r_2 \cdot n_g \rfloor + \delta$  ▷ Choose 1st node
17:    $r_3 \sim U[0, 1], j \leftarrow \lfloor r_3 \cdot (n_g - 1) \rfloor + \delta$  ▷ Choose 2nd node
18:   if  $j \geq i$  then
19:      $j \leftarrow j + 1$ 
20:   end if
21:   return  $(i, j)$ 

22: procedure BTER_SAMPLE_PHASE2( $\{w_d\}, \{r_d\}, \{n_d\}, \{n_d^{\text{fill}}\}, \{i_d\}$ )
23:    $i \leftarrow \text{BTER\_SAMPLE\_PHASE2\_NODE}(\{w_d\}, \{r_d\}, \{n_d\}, \{n_d^{\text{fill}}\}, \{i_d\})$ 
24:    $j \leftarrow \text{BTER\_SAMPLE\_PHASE2\_NODE}(\{w_d\}, \{r_d\}, \{n_d\}, \{n_d^{\text{fill}}\}, \{i_d\})$ 
25:   return  $(i, j)$ 

26: procedure BTER_SAMPLE_PHASE2_NODE( $\{w_d\}, \{r_d\}, \{n_d\}, \{n_d^{\text{fill}}\}, \{i_d\}$ )
27:    $d \leftarrow \text{RANDOM\_SAMPLE}(\{w_d\})$  ▷ Choose degree
28:    $r_1 \sim U[0, 1], r_2 \sim U[0, 1]$ 
29:   if  $r_1 < r_d^{\text{fill}}$  then
30:      $i \leftarrow \lfloor r_2 \cdot n_d^{\text{fill}} \rfloor + i_d$  ▷ Fill node
31:   else
32:      $i \leftarrow \lfloor r_2 \cdot (n_d - n_d^{\text{fill}}) \rfloor + (i_d + n_d^{\text{fill}})$  ▷ Bulk node
33:   end if
34:   return  $i$ 
```

---

In Algorithm 2, we generate each edge independently. It may also be possible to “bulk” the computations by first determining the total number of edges for each phase and perform the computation for each phase separately. Within each phase, the procedure itself can be easily vectorized to boost runtime performance, as in MATLAB.

**A.3. Edge Deduplication.** Any method can be used for deduplication. In general, the simplest procedure is to hash the edges in such a way that  $(i, j)$  and  $(j, i)$  hash to the same key. Then its easy enough to sort each bucket to remove duplicates. In a parallel environment, since we are hashing by edge and not vertex, there should not be load balancing problems. In fact, hashing by a single endpoint is

not recommended because of the heavy-tailed nature of the graph.

## Appendix B. Coupon Collector Derivation.

Consider a universe of  $U$  of objects/coupons, and suppose we pick objects uniformly at random with replacement from  $U$ . The following theorem proves the desired bound, when  $U$  is the set of possible pairs in an affinity block (so  $|U| = \binom{n_b}{2}$ ).

**THEOREM B.1.** *For a given  $\rho \in (0, 1)$ , the expected number of independent draws required to select  $\rho|U|$  distinct coupons from  $U$  is  $|U| \ln(1/(1 - \rho)) + O(1)$ .*

*Proof.* For convenience, we assume that  $\rho|U|$  is an integer. Consider a sequence of draws. Let  $X_i$  (for integer  $0 \leq i < \rho|U|$ ) be the random variable denoting the number of draws required to get one more (distinct) coupon after  $i$  distinct coupons have been collected. Observe that the quantity of interest is  $\mathbb{E}[\sum_{i < \rho|U|} X_i]$ , which by linearity of expectation is  $\sum_{i < \rho|U|} \mathbb{E}[X_i]$ .

When  $i$  distinct coupons have already been collected, the probability that a single draw gives a new coupon is exactly  $1 - i/|U|$ . Think of this as probability of “failure.” The number of draws required for a success (new coupon) follows a geometric distribution (Chap VI.8 of [16]) and the mean of this is  $1/(1 - i/|U|) = |U|/(|U| - i)$ . Using this bound, the expected total number of draws can be expressed as follows:

$$\begin{aligned} \sum_{i < \rho|U|} \mathbb{E}[X_i] &= \sum_{i < \rho|U|} \frac{|U|}{|U| - i} \\ &= |U| \left[ \sum_{i \leq |U|} \frac{1}{i} - \sum_{i \leq (1-\rho)|U|} \frac{1}{i} \right] \\ &= |U| \left[ \ln |U| - \ln((1 - \rho)|U|) + O(1/|U|) \right] = |U| \ln(1/(1 - \rho)) + O(1) \end{aligned}$$

(We use the standard bound for the Harmonic sum:  $\sum_{i \leq r} 1/i = \ln r + \gamma + O(1/r)$ , where  $\gamma$  is Euler-Mascheroni constant.)  $\square$

## REFERENCES

- [1] W. AIELLO, F. CHUNG, AND L. LU, *A random graph model for power law graphs*, Experimental Mathematics, 10 (2001), pp. 53–66.
- [2] A.-L. BARABÁSI AND R. ALBERT, *Emergence of scaling in random networks*, Science, 286 (1999), pp. 509–512.
- [3] A. BARRAT AND M. WEIGT, *On the properties of small-world network models*, The European Physical Journal B - Condensed Matter and Complex Systems, 13 (2000), pp. 547–560.
- [4] Z. BI, C. FALOUTSOS, AND F. KORN, *The “DGX” distribution for mining massive, skewed data*, in KDD ’01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2001, pp. 17–26.
- [5] P. BOLDI, M. ROSA, M. SANTINI, AND S. VIGNA, *Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks*, in WWW’11: Proceedings of the 20th international conference on World Wide Web, ACM Press, 2011.
- [6] P. BOLDI, M. SANTINI, AND S. VIGNA, *A large time-aware graph*, SIGIR Forum, 42 (2008), pp. 33–38.
- [7] P. BOLDI AND S. VIGNA, *The webgraph framework I: Compression techniques*, in WWW’04: Proceedings of the 13th International Conference on World Wide Web, ACM Press, 2004, pp. 595–602.
- [8] D. CHAKRABARTI, Y. ZHAN, AND C. FALOUTSOS, *R-MAT: A recursive model for graph mining*, in SDM04: Proceedings of the Fourth SIAM International Conference on Data Mining, April 2004.
- [9] F. CHIERICETTI, R. KUMAR, S. LATTANZI, M. MITZENMACHER, A. PANCONESI, AND P. RAGHAVAN, *On compressing social networks*, in KDD ’09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2009, pp. 219–228.

- [10] F. CHUNG AND L. LU, *The average distances in random graphs with given expected degrees*, Proceedings of the National Academy of Sciences, 99 (2002), pp. 15879–15882.
- [11] F. CHUNG AND L. LU, *Connected components in random graphs with given degree sequences*, Annals of Combinatorics, 6 (2002), pp. 125–145.
- [12] A. CLAUSET, C. R. SHALIZI, AND M. E. J. NEWMAN, *Power-law distributions in empirical data*, SIAM Review, 51 (2009), pp. 661–703.
- [13] D. DOMINGUEZ-SAL, P. URBÓN-BAYES, A. GIMÉNEZ-VANÓ, S. GÓMEZ-VILLAMOR, N. MARTNEZ-BAZÁN, AND J. LARRIBA-PEY, *Survey of graph database performance on the HPC scalable graph analysis benchmark*, in Web-Age Information Management, H. Shen, J. Pei, M. Özsu, L. Zou, J. Lu, T.-W. Ling, G. Yu, Y. Zhuang, and J. Shao, eds., vol. 6185 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2010, pp. 37–48.
- [14] N. DURAK, T. G. KOLDA, A. PINAR, AND C. SESHADHRI, *A scalable directed graph model with reciprocal edges*, in Proceedings of the IEEE 2nd Workshop on Network Science, 2013. to appear, also available as arXiv:1210.5288.
- [15] P. ERDÖS AND A. RÉNYI, *On the evolution of random graphs*, Publication of the Mathematical Institute of the Hungarian Academy Of Sciences, (1960).
- [16] W. FELLER, *An Introduction to probability theory and applications: Vol I*, John Wiley and Sons, 3rd ed., 1968.
- [17] M. GIRVAN AND M. E. J. NEWMAN, *Community structure in social and biological networks*, Proceedings of the National Academy of Sciences, 99 (2002), pp. 7821–7826.
- [18] D. F. GLEICH AND A. B. OWEN, *Moment-based estimation of stochastic Kronecker graph parameters*, Internet Mathematics, 8 (2012), pp. 232–256.
- [19] W. GUO AND S. KRAINES, *A random network generator with finely tunable clustering coefficient for small-world social networks*, in CASON '09: International Conference on Computational Aspects of Social Networks, June 2009, pp. 10–17.
- [20] T. G. KOLDA, A. PINAR, T. PLANTENGA, C. SESHADHRI, AND C. TASK, *Counting triangles in massive graphs with MapReduce*. arXiv:1301.5887, Jan. 2013.
- [21] R. KUMAR, P. RAGHAVAN, S. RAJAGOPALAN, D. SIVAKUMAR, A. TOMKINS, AND E. UPFAL, *Stochastic models for the web graph*, in Proceedings of 41st Annual Symposium on Foundations of Computer Science, 2000, pp. 57–65.
- [22] H. KWAK, C. LEE, H. PARK, AND S. MOON, *What is Twitter, a social network or a news media?*, in WWW '10: Proceedings of the 19th international conference on World wide web, New York, NY, USA, 2010, ACM, pp. 591–600.
- [23] J. LESKOVEC, D. CHAKRABARTI, J. KLEINBERG, C. FALOUTSOS, AND Z. GHAHRAMANI, *Kronecker graphs: An approach to modeling networks*, Journal of Machine Learning Research, 11 (2010), pp. 985–1042.
- [24] J. LESKOVEC, J. KLEINBERG, AND C. FALOUTSOS, *Graph evolution: Densification and shrinking diameters*, ACM Transactions on Knowledge Discovery from Data, 1 (2007). Article No. 2.
- [25] M. MIHAIL AND C. PAPADIMITRIOU, *On the eigenvalue power law*, in RANDOM 2002: Proceedings of Randomization and Approximation Techniques in Computer Science, vol. 2483 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2002, pp. 254–262.
- [26] B. MILLER, N. BLISS, AND P. WOLFE, *Subgraph detection using eigenvector L1 norms*, in NIPS 2010: Advances in Neural Information Processing Systems 23, 2010, pp. 1633–1641.
- [27] B. MILLER, L. STEPHENS, AND N. BLISS, *Goodness-of-fit statistics for anomaly detection in Chung-Lu random graphs*, in ICASSP 2012: IEEE International Conference on Acoustics, Speech and Signal Processing, march 2012, pp. 3265–3268.
- [28] D. MIR AND R. N. WRIGHT, *A differentially private estimator for the stochastic Kronecker graph model*, in EDBT-ICDT '12: Proceedings of the 2012 Joint EDBT/ICDT Workshops, ACM, 2012, pp. 167–176.
- [29] S. MORENO, S. KIRSHNER, J. NEVILLE, AND S. V. N. VISHWANATHAN, *Tied Kronecker product graph models to capture variance in network populations*, in Proceedings of 2010 48th Annual Allerton Conference on Communication, Control, and Computing, Oct. 2010, pp. 1137–1144.
- [30] M. NEWMAN, D. WATTS, AND S. STROGATZ, *Random graph models of social networks*, Proceedings of the National Academy of Sciences, 99 (2002), pp. 2566–2572.
- [31] M. E. J. NEWMAN, *Properties of highly clustered networks*, Phys. Rev. E, 68 (2003), p. 026121.
- [32] M. E. J. NEWMAN, *Finding community structure in networks using the eigenvectors of matrices*, Physical Review E, 74 (2006), p. 036104.
- [33] A. PINAR, C. SESHADHRI, AND T. G. KOLDA, *The similarity between stochastic Kronecker and Chung-Lu graph models*, in SDM12: Proceedings of the Twelfth SIAM International Conference on Data Mining, Apr. 2012, pp. 1071–1082.
- [34] A. SALA, L. CAO, C. WILSON, R. ZABLIT, H. ZHENG, AND B. Y. ZHAO, *Measurement-calibrated*



- graph models for social network experiments*, in WWW '10: Proceedings of the 19th international conference on World wide web, 2010, pp. 861–870.
- [35] A. SALA, S. GAITO, G. P. ROSSI, H. ZHENG, AND B. Y. ZHAO, *Revisiting degree distribution models for social graph analysis*. arXiv:1108.0027, July 2011.
  - [36] C. SESHADHRI, T. G. KOLDA, AND A. PINAR, *Community structure and scale-free collections of Erdős-Rényi graphs*, Physical Review E, 85 (2012), p. 056109.
  - [37] C. SESHADHRI, A. PINAR, AND T. G. KOLDA, *An in-depth study of stochastic Kronecker graphs*, in ICDM 2011: Proceedings of the 2011 IEEE International Conference on Data Mining, 2011, pp. 587–596.
  - [38] —, *Triadic measures on graphs: The power of wedge sampling*. arXiv:1202.5230 [cs.SI], October 2012. to appear in SDM13.
  - [39] —, *An in-depth analysis of stochastic Kronecker graphs*, Journal of the Association for Computing Machinery, in press (2013). Preprint: <http://arxiv.org/abs/1102.5046>.
  - [40] J. C. VIVAR AND D. BANKS, *Models for networks: a cross-disciplinary science*, Wiley Interdisciplinary Reviews: Computational Statistics, 4 (2012), pp. 13–27.
  - [41] D. WATTS AND S. STROGATZ, *Collective dynamics of ‘small-world’ networks*, Nature, 393 (1998), pp. 440–442.
  - [42] *Graph 500 benchmark*. Available at <http://www.graph500.org/Specifications.html>.
  - [43] *LWA: Laboratory for Web Algorithms*. Available at <http://law.di.unimi.it/datasets.php>.
  - [44] *SNAP: Stanford Network Analysis Project*. Available at <http://snap.stanford.edu/>.